图文 066、阶段性复习：JVM运行原理和GC原理你真的搞懂了吗？



详情 评论

**阶段性复习：**
**JVM运行原理和GC原理你真的搞懂了吗？**

---

狸猫技术窝专栏上新，基于**真实订单系统**的消息中间件（mq）实战，重磅推荐：


**未来3个月，我的好朋友原子弹大侠将带你一起，全程实战，360度死磕MQ**

（点击下方蓝字进行试听）

**从 0 开始带你成为消息中间件实战高手**

---

**重要说明：**

**如何提问**：每篇文章都有评论区，大家可以尽情在评论区留言提问，我都会逐一答疑

（**ps**：评论区还精选了一些小伙伴对**专栏每日思考题的作答**，有的答案真的非常好！大家可以通过看别人的思路，启发一下自己，从而加深理解）

**如何加群**：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**。

（群里有不少**一二线互联网大厂的助教**，大家可以一起讨论交流各种技术）

具体**加群方式**请参见文末。

（**注**：以前通过其他专栏加过群的同学就不要重复加了）

---

**1、阶段性复习**

最近三天会进行一个阶段性的复习，因为我们已经把完整的JVM运行原理、GC原理以及GC优化的原理，还有线上发生GC问题的各种优化案例，都给大家完整的分析完了，所以到这里务必停一停脚步，整理一下学习过的知识脉络，让大家进行一点复习。反复的复习，才能让大家真正吃透和消化掉这些知识。

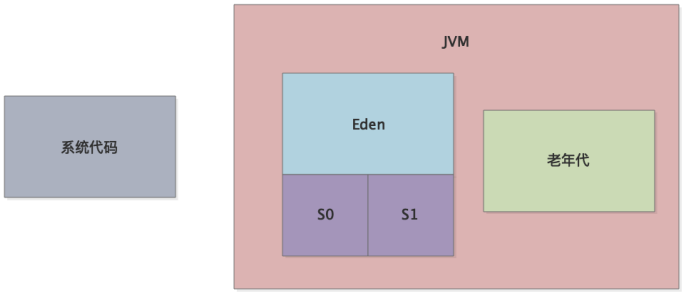
这几篇文章我们不会把之前文章的东西大幅度的整理出来，主要是给大家提点一些思路，给你一些引导，希望每个人看到这里的时候，都停一停脚步，顺着我们带出来的思路，把过往学习过的知识，自己整理一下。

**2、JVM和GC的运行原理，你都能搞懂了吗?**

对于JVM的学习，首先大家务必要搞清楚一点，JVM是如何运行起来的。

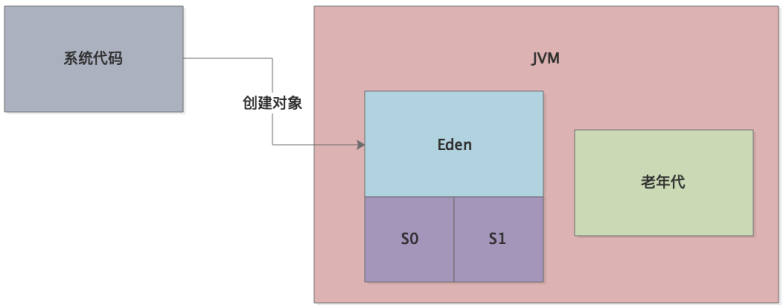
相信大家认真看过之前文章的，应该都清楚一点，JVM的内存区域划分，最核心的就是这么几块了：年轻代、老年代、Metaspace（也就是以前的永久代）。

其中年轻代又分成了Eden和2个Survivor，默认比例是8：1：1，如下图。



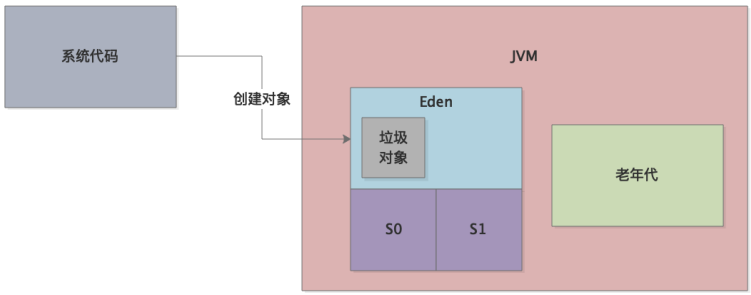

接着我们来思考一下，我们写好的系统会不停的运行，运行的时候是不是就会不停的在年轻代的Eden区域中创建各种对象?

如下图所示。




而且一般创建对象都是在各种方法里执行的，一旦方法运行完毕，方法局部变量引用的那些对象就会成为Eden区里的垃圾对象，就是可以被回收的状态，大家务必要清楚这个过程

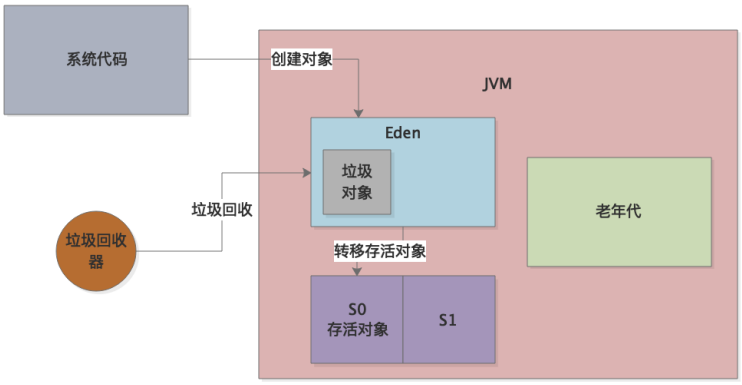
如下图。




接着随着Eden区不断的创建对象，就会逐步的塞满，当然这个时候可能塞满Eden区的对象里大多数都是垃圾对象。一旦Eden区塞满之后，就会触发一次Young GC。

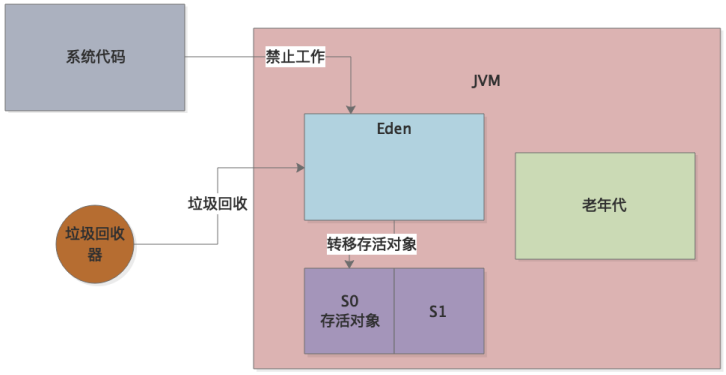
Young GC会采用复制算法，从GC Roots（方法的局部变量、类的静态变量）开始追踪，标记出来存活的对象。

然后把存活对象都放入第一个Survivor区域中，也就是S0区域，如下图所示。

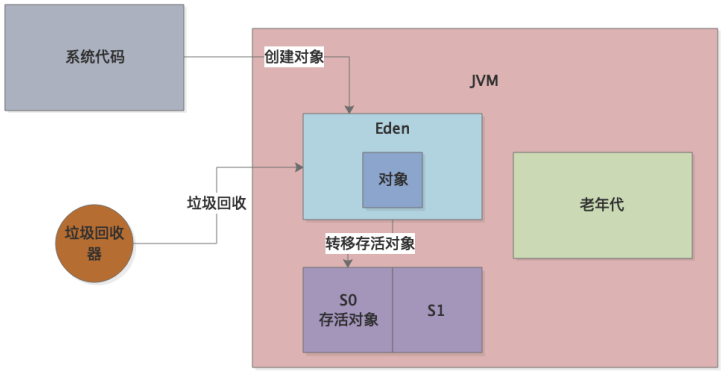



接着垃圾回收器就会直接回收掉Eden区里剩余的全部垃圾对象，在整个这个垃圾回收的过程中全程会进入Stop the Wold状态，也就是暂停系统工作线程，系统代码全部停止运行，不允许创建新的对象

只有这样，才能让垃圾回收器专心工作，找出来存活对象，回收掉垃圾对象，如下图所示。

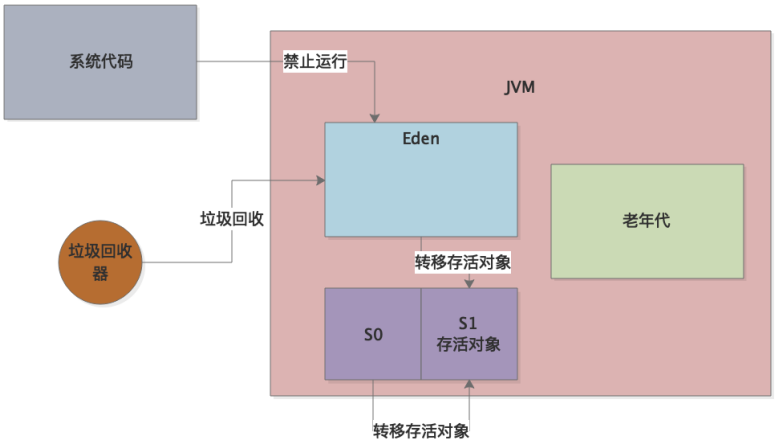



一旦垃圾回收全部完毕之后，也就是存活对象都进入了Survivor区域，然后Eden区都清空了，那么Young GC执行完毕，此时系统恢复工作，继续在Eden区里创建对象，如下图所示。




下一次如果Eden区满了，就会再次触发Young GC，把Eden区和S0区里的存活对象转移到S1区里去，然后直接清空掉Eden区和S0区中的垃圾对象

当然这个过程中系统是禁止运行的，处于Stop the World状态，如下图所示。




负责Young GC的垃圾回收器有很多种，但是常用的就是ParNew垃圾回收器，他的核心执行原理就如上所述，只不过他运行的时候是基于多线程并发执行垃圾回收的，大家只要记得这点就可以。

这就是最基本的JVM和GC的运行原理，大家都搞懂了吗？

**3、对象什么时候进入老年代？**

但是大家觉得光是一块年轻代和Young GC配合起来，就足够JVM来使用了吗？

No！

实际JVM运行过程中，有很多意外的情况会发生的，会导致对象进入老年代区域中，如下所述几种情况，反复给大家总结过，务必要记得很清晰：

一个对象在年轻代里躲过15次垃圾回收，年龄太大了，寿终正寝，进入老年代

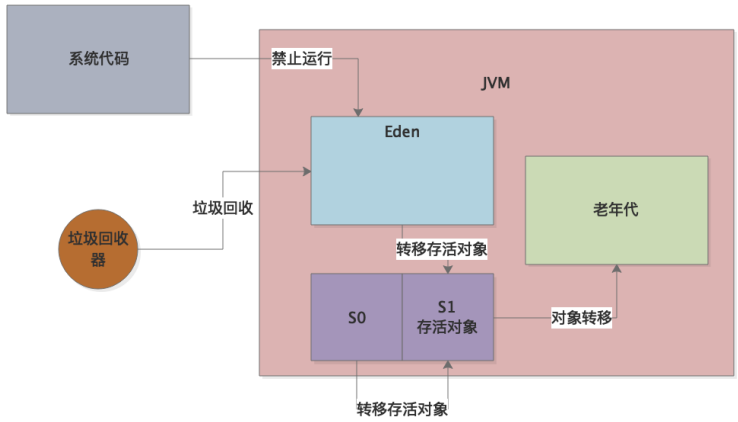
对象太大了，超过了一定的阈值，直接进入老年代，不走年轻代

一次Young GC过后存活对象太多了，导致Survivor区域放不下了，这批对象会进入老年代

可能几次Young GC过后，Surviovr区域中的对象占用了超过50%的内存，此时会判断如果年龄1+年龄2+年龄N的对象总和超过了Survivor区域的50%，此时年龄N以及之上的对象都进入老年代，这是动态年龄判定规则

上面4个条件就是最常见的对象进入老年代的情况，那种长期存活的躲过15次Young GC的对象毕竟是少数的，大对象一般在特殊情况下会有，对于那种加载大量数据长时间处理以及高并发的场景，很容易导致Young GC后存活对象过多的。

所以对于这些情况，都会导致对象进入老年代中，老年代对象可能会越来越多，如下图所示。




**4、老年代的GC是如何触发的?**

一旦老年代对象过多，就可能会触发Full GC，Full GC必然会带着Old GC，也就是针对老年代的GC

而且一般会跟着一次Young GC，也会触发永久代的GC。

大家还记得Full GC触发的几个条件吗?

老年代自身可以设置一个阈值，有一个JVM参数可以控制，一旦老年代内存使用达到这个阈值，就会触发Full GC，一般建议调节大一些，比如92%

在执行Young GC之前，如果判断发现老年代可用空间小于了历次Young GC后升入老年代的平均对象大小的话，那么就会在Young GC之前触发Full GC，先回收掉老年代一批对象，然后再执行Young GC。

如果Young GC过后的存活对象太多，Survivor区域放不下，就要放入老年代，要是此时老年代也放不下，就会触发Full GC，回收老年代一批对象，再把这些年轻代的存活对象放入老年代中

触发Full GC几个比较核心的条件就是这几个，总结起来，其实就是老年代一旦快要搞满了，空间不够了，必然要垃圾回收一次。

老年代的垃圾回收通常建议走CMS垃圾回收器，回收机制比较复杂，此处建议大家自行复兴和总结一下

总之，Old GC的速度是很慢的，少则几百毫秒，多则几秒。所以一旦Full GC很频繁，就会导致系统性能很差，因为频繁要停止系统工作线程，导致系统看起来一直有卡顿的现象。

而且频繁Full GC还会导致机器CPU负载过高，导致机器性能下降，处理请求能力降低。

所以优化JVM的核心就是减少Full GC的频率。

**5、正常情况下的系统**

正常情况下的系统，会有一定频率的Young GC，一般在几分钟一次Young GC，或者几十分钟一次Young GC，一次耗时在几毫秒到几十毫秒的样子，都是正常的。

正常的Full GC频率在几十分钟一次，或者几个小时一次，这个范围内都是正常的，一次耗时应该在几百毫秒的样子。

所以大家如果观察自己线上系统就是这个性能表现，基本上问题都不太大。

当然，实际线上系统很多时候回遇到一些JVM性能问题，就是Full GC过于频繁，每次还耗时很多的情况，此时就需要一些优化了。

明天我们会继续总结Full GC优化的一些常用手段以及生产环境下的GC优化的方法。

**End**